Sadržaj

Predgovor ..... **xiii**

1. Uvod u JavaScript **1**
1.1 Istraživanje JavaScripta 3
1.2 Hello World 4
1.3 Tura kroz JavaScript 5
1.4 Primer: Histogrami učestalosti znakova 11
1.5 Rezime 13

2. Leksička struktura **15**
2.1 Tekst JavaScript programa 15
2.2 Komentari 16
2.3 Literali 16
2.4 Identifikatori i rezervisane reči 16
2.5 Unicode 17
2.6 Neobavezni znaci tačka sa zapetom 18
2.7 Rezime 20

3. Tipovi, vrednosti i promenljive **21**
3.1 Pregled i definicije 21
3.2 Brojevi 23
3.3 Tekst 29
3.4 Bulove vrednosti 35
3.5 null i undefined 36
3.6 Symbol 37
3.7 Globalni objekat 38
3.8 Nepromenljive osnovne vrednosti i reference promenljivih objekata 39
3.9 Konverzije tipa 41
3.10 Deklaracija promenljive i dodeljivanje vrednosti 47
3.11 Rezime 54

4. Izrazi i operatori 55
4.1 Primarni izrazi 55
4.2 Inicijalizatori objekata i nizova 56
4.3 Izrazi definisanja funkcije 57
4.4 Izrazi pristupa svojstvu 58
4.5 Izrazi poziva 59
4.6 Izrazi stvaranja objekta 61
4.7 Pregled operatora 62
4.8 Aritmetički izrazi 66
4.9 Relacioni izrazi 70
4.10 Logički izrazi 75
4.11 Izrazi dodele 77
4.12 Izrazi evaluacije 79
4.13 Raznovrsni operatori 82
4.14 Rezime 86

5. Naredbe 87
5.1 Naredbe izraza 88
5.2 Složene i prazne naredbe 88
5.3 Uslovi 89
5.4 Petlje 94
5.5 Skokovi 100
5.6 Ostale naredbe 107
5.7 Deklaracije 110
5.8 Rezime JavaScript naredbi 113

6. Objekti 115
6.1 Uvod u objekte 115
6.2 Izrada objekata 116
6.3 Pretraživanje i postavljanje svojstava 119
6.4 Brisanje svojstava 123
6.5 Ispitivanje svojstava 124
6.6 Svojstva nabrajanja 125
6.7 Proširivanje objekata 126
6.8 Serijalizacija objekata 127
6.9 Metode objekta 128
6.10 Proširena sintaksa literala objekta 130
6.11 Rezime 136

7. Nizovi 137
7.1 Stvaranje nizova 138
7.2 Čitanje i upisivanje elemenata niza 140
7.3 Proređeni nizovi 141
7.4 Dužina niza 142

7.5 Dodavanje i brisanje elemenata niza 143
7.6 Iteracija nizova 143
7.7 Višedimenzionalni nizovi 145
7.8 Metode niza 145
7.9 Objekti slični nizu 156
7.10 Znakovni nizovi kao nizovi 158
7.11 Rezime 159

8. Funkcije 161
8.1 Definisanje funkcija 161
8.2 Funkcije pozivanja 165
8.3 Argumenti i parametri funkcije 171
8.4 Funkcije kao vrednosti 178
8.5 Funkcije kao imenski prostor 180
8.6 Zatvaranja 181
8.7 Svojstva, metode i konstruktor funkcije 186
8.8 Funkcionalno programiranje 190
8.9 Rezime 195

9. Klase 197
9.1 Klase i prototipovi 198
9.2 Klase i konstruktori 199
9.3 Klase sa rezervisanom rečju class 204
9.4 Dodavanje metoda postojećim klasama 210
9.5 Potklase 211
9.6 Rezime 222

10. Moduli 223
10.1 Moduli sa klasama, objektima i zatvaranjima 223
10.2 Moduli u Nodu 226
10.3 Moduli u ES6 228
10.4 Rezime 238

11. Standardna biblioteka JavaScripta 239
11.1 Setovi i mape 240
11.2 Tipski nizovi i binarni podaci 246
11.3 Poklapanje obrazaca pomoću regularnih izraza 251
11.4 Date i Time 269
11.5 Klase Error 273
11.6 JSON Serijalizacija i rašlanjivanje 274
11.7 API internacionalizacija 277
11.8 API konzole 284
11.9 URL API-ja 287
11.10 Tajmeri 290
11.11 Rezime 291

12. Iteratori i generatori 293
12.1 Kako rade iteratori 294
12.2 Primena iterabilnih objekata 294
12.3 Generatori 298
12.4 Napredne osobine generatora 301
12.5 Rezime 304

13. Asinhroni JavaScript 307
13.1 Asinhrono programiranje sa povratnim pozivima 307
13.2 Obećanja 311
13.3 async i await 331
13.4 Asinhrone iteracije 333
13.5 Rezime 339

14. Metaprogramiranje 341
14.1 Atributi svojstva 341
14.2 Proširivost objekta 346
14.3 Atributi prototipa 347
14.4 Dobro poznati simboli 348
14.5 Oznake obrazaca 355
14.6 Reflect API 357
14.7 Proxy objekti 359
14.8 Rezime 365

15. JavaScript u veb čitačima 367
15.1 Osnove veb programiranja 369
15.2 Događaji 382
15.3 Skriptiovanje dokumenata 392
15.4 CSS stvoren JavaScriptom 405
15.5 Geometrija i skrolovanje dokumenata
15.6 Veb komponente 415
15.7 SVG: Skalabilna vektorska grafika 427
15.8 Grafika u elementu <canvas> 434
15.9 Audio API 454
15.10 Lokacija, navigacija i istorija 456
15.11 Umrežavanje 464
15.12 Skladištenje 480
15.13 Radne niti i slanje poruka 491
15.14 Primer: Mandelbrot set 498
15.15 Rezime i predlozi za dalje čitanje 510

16. JavaScript na strani servera sa Nodom 517
16.1 Osnove Node programiranja 518

16.2 Node je podrazumevano asinhron 522
16.3 Baferi 526
16.4 Događaji i EventEmitter 527
16.5 Protoci 529
16.6 Pojedinosti o Processu, CPU-u i operativnom sistemu 539
16.7 Rad sa datotekama 541
16.8 HTTP klijenti i serveri 550
16.9 Mrežni serveri i klijenti koji nisu HTTP 555
16.10 Rad sa procesima Child 557
16.11 Niti Worker 562
16.12 Rezime 570

17. JavaScript alati i proširenja 571
17.1 Povezivanje sa ESLintom 572
17.2 JavaScript formatiranje sa altom Prettier 572
17.3 Ispitivanje jedinice pomoću Jesta 573
17.4 Upravljanje paketom sa npm 576
17.5 Paket kodova 577
17.6 Transpilacija sa Babelom 579
17.7 JSX: Oznake izraza u JavaScriptu 580
17.8 Provera tipa sa Flow 584
17.9 Rezime 598

Rečnik termina korišćenih u knjizi 599

Indeks 601

Predgovor

Ova knjiga pokriva JavaScript jezik i JavaScript API-je implementirane od strane veb čitača i Noda. Napisao sam je čitaocima sa određenim prethodnim iskustvom u programiranju koji žele da nauče JavaScript, ali i programerima koji već koriste JavaScript, ali žele da svoje razumevanje podignu na viši nivo i zaista savladaju jezik. Moj cilj sa ovom knjigom je da dokumentujem JavaScript jezik sveobuhvatno i kompletno i pružim dubinski uvod u najvažnije API-je na strani klijenta i na strani servera koji su danas dostupni. Kao rezultat, ovo je obimna i detaljna knjiga. Ali nadam se da će pažljivo proučavanje biti nagrađeno i da će vam vreme koje potrošite da pročitate knjigu, biti nadoknađeno u vidu vaše veće produktivnosti u programiranja.

Prethodna izdanja ove knjige sadržala su opsežan referentni deo. Više ne smatram da ima smisla da taj materijal bude u štampanom obliku kada se na mreži mogu lako i brzo naći ažurni referentni materijali. Ako treba da nađete bilo šta u vezi sa jezgrom ili klijentskom stranom JavaScripta, preporučujem vam da posetite MDN veb lokaciju (*https://developer.mozilla.org)*. A za Node API-je na strani servera, preporučujem vam da odete direktno na izvor i obratite se referentnoj dokumentaciji (*https://nodejs.org/api*).

Konvencije korišćene u ovoj knjizi

U ovoj knjizi koristim sledeće tipografske konvencije:

Kurziv (Italic)
: Koristi se za naglašavanje i označavanje prve upotrebe termina. *Kurziv* se koristi i za adrese e-pošte, URL adrese i imena datoteka.

`Font jednake širine (Constant width)`
: Koristi se u svim JavaScript kodovima i CSS i HTML listinzima, i uopšte za sve što biste doslovno kucali prilikom programiranja.

`Kurziv jednake širine (Constant width italic)`
: Povremeno se koristi kada objašnjava sintaksu JavaScripta.

`Podebljano jednake širine`
: Prikazuje naredbe ili drugi tekst koji korisnik treba doslovno da upiše.



Ovaj znak označava opštu napomenu.



Ovaj znak ukazuje na upozorenje ili oprez.

Primer koda

Dopunski materijal (primeri koda, vežbe, itd.) za ovu knjigu dostupan je za preuzimanje na:

https://oreil.ly/javascript_defgd7

Ova knjiga je tu da vam pomogne da uradite svoj posao. Generalno, ako se uz ovu knjigu nudi primer koda, možete ga koristiti u svojim programima i dokumentaciji. Ne morate nas kontaktirati za dozvolu ako ne reprodukujete značajan deo koda. Na primer, za pisanje programa koji koristi nekoliko delova koda iz ove knjige nije potrebno odobrenje. Za prodaju ili distribuciju primera iz *O'Reilly/Mikro knjga* knjige potrebna je dozvola. Da biste odgovorili na pitanje citirajući ovu knjigu i navodeći primer koda, ne zahteva dozvolu. Uključivanje značajne količine primera koda iz ove knjige u dokumentaciju vašeg proizvoda zahteva dozvolu.

Generalno ne zahtevamo atribuciju, ali ćemo je ceniti. Atribucija obično sadrži: naslov, autora, izdavača i ISBN. Na primer: „*JavaScript: Sveobuhvatan vodič*, sedmo izdanje, autor David Flanagan (Mikro knjiga/O'Reilli). Copyright 2020 David Flanagan, 978-1-491-95202-3, Copyright 2020 Mikro knjiga, 978-86-7555-448-6"

Za prava preizimanja koda pišite na `permissions@oreilli.com` ili *redakcija@mikroknjiga.rs*

O'Reilly online učenje

O'Reilly Media već više od 40 godina pruža tehnologiju i poslovnu obuku, znanja i uvide da bi kompanijama pomogla da uspešno posluju.

Naša jedinstvena mreža stručnjaka i inovatora deli svoje znanje i stručnost kroz knjige, članke i našu platformu za online učenje. O'Reilly platforma za učenje na mreži pruža vam pristup zahtevima za obuku uživo, detaljnim načinima učenja, interaktivnim okruženjima za kodiranje i ogromnu kolekciju teksta i video zapisa od O'Reillya i više od 200 drugih izdavača. Za više informacija posetite `http://oreilly.com`.

Izdanja Mikro knjige

Mikro knjiga od 1984. godine pruža najbolje naslove iz oblasti računske tehnike i informatike. Sa preko 450 objavljenih naslova pomogla je mnogim pojedincima, preduzećima i ustanovama da se obrazuju i uspešno posluju. Posebna pažnja je poklonjena u izboru domaćih i stranih autora, kvalitetu prevoda, svebuhvatnih indeksa i spiskova termnina koji su sadržani u našim knjigama. Kao rezultat tih napora na vebu se nalazi svima dostupan *Računarski rečnik Mikro knjige* (*http://www.mikroknjiga.rs/recnik*).

Kako da nas kontaktirate

Molimo da komentare i pitanja koja se odnose na ovu knjigu pošaljete izdavaču:

Mikro knjiga

P.fah 20-87
11030 Beograd, Srbija
Tel. 011 3055048 (redakcija)
Tel. 011 3055010 (prodaja)
redakcija@mikroknjiga.rs

Zahvalnice

Mnogi ljudi su pomogli u stvaranju ove knjige. Želeo bih da se zahvalim svojoj urednici Angeli Rufino što me drži u koloseku i što ima strpljenja za moje prekoračene rokove. Hvala i mojim tehničkim recenzentima: Brian Sletten, Elisabeth Robson, Ethan Flanagan, Maximiliano Firtman, Sarah Wachs i Schalk Neethling. Njihovi komentari i sugestije učinili su ovu knjigu boljom.

Proizvodni tim u O'Reillya odradio je svoj uobičajeni dobar posao: Kristen Brown je vodila proces proizvodnje, Deborah Baker je bila proizvodna urednica, Rebecca Demarest je nacrtala slike, a Judy McConville je stvorila indeks.

Urednici, recenzenti i saradnici prethodnih izdanja ove knjige su: Andrew Schulman, Angelo Sirigos, Aristotle Pagaltzis, Brendan Eich, Christian Heilmann, Dan Shafer, Dave C. Mitchell, Deb Cameron, Douglas Crockford, Dr. Tankred Hirschmann, Dylan Schiemann, Frank Willison, Geoff Stearns, Herman Venter, Jay Hodges, Jeff Yates, Joseph Kesselman, Ken Cooper, Larry Sullivan, Lynn Rollins, Neil Berkman, Mike Loukides, Nick Thompson, Norris Boyd, Paula Ferguson, Peter-Paul Koch, Philippe Le Hegaret, Raffaele Cecco, Richard Yaker, Sanders Kleinfeld, Scott Furman, Scott Isaacs, Shon Katzenberger, Terry Allen, Todd Ditchendorf, Vidur Apparao, Waldemar Horwat i Zachary Kessin.

Pisanje ovog sedmog izdanja udaljilo me od porodice mnogo kasnih noći. Moja ljubav i zahvalnost što su imali razumevanja za moje odsustvo.

– David Flanagan, mart 2020.

POGLAVLJE 1

Uvod u JavaScript

JavaScript je programski jezik veba. Velika većina veb lokacija koristi JavaScript, a svi savremeni veb čitači – na radnim površinama, tabletima i telefonima – uključuju JavaScript interpreter, čime je JavaScript najrasprostranjeniji programski jezik u istoriji. Tokom poslednje decenije, Node.js je omogućio JavaScript programiranje izvan veb čitača, a dramatičan uspeh Noda znači da je JavaScript sada i najkorišćeniji programski jezik među programerima softvera. Bez obzira da li počinjete ispočetka ili već profesionalno koristite JavaScript, ova knjiga će vam pomoći da savladate jezik.

Ako ste već upoznati sa drugim programskim jezicima, možda će vam pomoći da znate da je JavaScript visoko dinamični, interpretirani programski jezik koji je dobro prilagođen objektno orijentisanim i funkcionalnim stilovima programiranja. JavaScript promenljive nisu tipizirane. Njegova sintaksa je bazirana na Javi, ali jezici inače nisu povezani. JavaScript svoje prvoklasne funkcije dobija iz Sheme i nasleđivanja prototipa iz malo poznatog jezika Self. Ali ne treba da znate nijedan od tih jezika ili da budete upoznati sa tim terminima da biste koristili ovu knjigu i naučili JavaScript.

Naziv „JavaScript“ je prilično pogrešan. Osim površne sintaktičke sličnosti, JavaScript se u potpunosti razlikuje od Java programskog jezika. A JavaScript je odavno prerastao svoje poreklo skriptnog jezika i postao je robustan i efikasan jezik opšte namene pogodan za ozbiljan softverski inženjering i projekte sa ogromnim bazama podataka.

JavaScript: Imena, verzije i načini rada

JavaScript je nastao u Netscapu u ranim danima veba, a tehnički je „JavaScript“ zaštitni znak licenciran od strane Sun Microsystems (sada Oracle) koji se koristi da opiše implementaciju Netscapovog jezika (sada Mozille). Netscape je predao jezik na standardizaciju Evropskom udruženju proizvođača računara (ECMA) – i zbog problema sa zaštitnim znakom, standardizovana verzija jezika bila je zaglavljena s rogobatnim nazivom „ECMAScript“. U praksi, svi jezik zovu JavaScript. Ova knjiga koristi naziv „ECMAScript“ i skraćenicu „ES“ da bi se odnosila na jezički standard i na verzije tog standarda.

Za veći deo 2010-ih, verziju 5 ECMAScript standarda podržavaju svi veb čitači. Ova knjiga tretira ES5 kao osnovu kompatibilnosti i više ne raspravlja o starijim verzijama jezika. ES6 je objavljen 2015. godine i dodao je nove funkcije – uključujući sintaksu klase i modula – koje su JavaScript promenile iz skriptnog jezika u ozbiljan opšti namenski jezik pogodan za softverski inženjering velikih razmera. Otkad je ES6, ECMAScript specifikacija prešla na godišnja izdanja, verzije jezika – ES2016, ES2017, ES2018, ES2019 i ES2020 – sada su identifikovane prema godini izdavanja.

Kako se JavaScript razvijao, dizajneri jezika pokušali su da isprave nedostatke u ranim (pre-ES5) verzijama. Da bi se održala kompatibilnost unazad, nije moguće ukloniti nasleđene funkcije, bez obzira koliko bile pogrešne. Ali u ES5 i novijim programima, programi mogu da se uključe u strogi JavaScript režim (*strict mode*) u kome su ispravljene brojne ranije greške jezika. Mehanizam za prijavu je direktiva „stroga upotreba" opisana u §5.6.3. U ovom odeljku sumirane su i razlike između starijeg i strogog JavaScripta. U ES6 i novijim, upotreba novih jezičnih funkcija često implicitno poziva na strogi režim rada. Na primer, ako koristite ključnu reč `class` u ES6 ili stvarate ES6 modul, tada je sav kod unutar klase ili modul je automatski strog, a stare, pogrešne funkcije nisu dostupne u tim kontekstima. Ova knjiga će pokriti nasleđene funkcije JavaScripta, ali pažljivo će naglasiti da one nisu dostupne u strogom režimu.

Da bi bio koristan, svaki jezik mora imati platformu ili standardnu biblioteku za obavljanje stvari kao što su osnovni unos i izlaz. Jezgro JavaScript jezike definiše minimalan API za rad sa brojevima, tekstom, nizovima, skupovima, mapama i slično, ali ne uključuje bilo koju ulaznu ili izlaznu funkcionalnost. Ulaz i izlaz (kao i sofisticiranije funkcije, kao što su rad sa mrežom, skladištenje i grafika) su odgovornost „domaćina" (engl. *host environment*) u okviru kojeg je ugrađen JavaScript.

Prvobitno okruženje domaćina za JavaScript bio je veb čitač, a to je i dalje najčešće okruženje u kome se izvršava JavaScript kod. Okruženje veb čitača omogućava JavaScript kod da dobije unos iz korisnikovog miša i tastature i pomoću HTTP zahteva. I omogućava JavaScript kodu da prikazuje izlaz korisniku korišćenjem HTML-a i CSS-a.

Od 2010. godine za JavaScript kod je dostupno još jedno okruženje domaćina. Umesto ograničavanja JavaScripta da radi sa API-jima koje pruža veb pretraživač, Node omogućava JavaScript pristup celom operativnom sistemu, omogućavajući JavaScript programima da čitaju i pišu datoteke, šalju i primaju podatke preko mreže i prave i opslužuju HTTP zahteve. Node je popularan izbor za implementaciju veb servera i takođe zgodan alat za pisanje jednostavnih uslužnih skripti kao alternativa skriptu školjke (engl. *shell*).

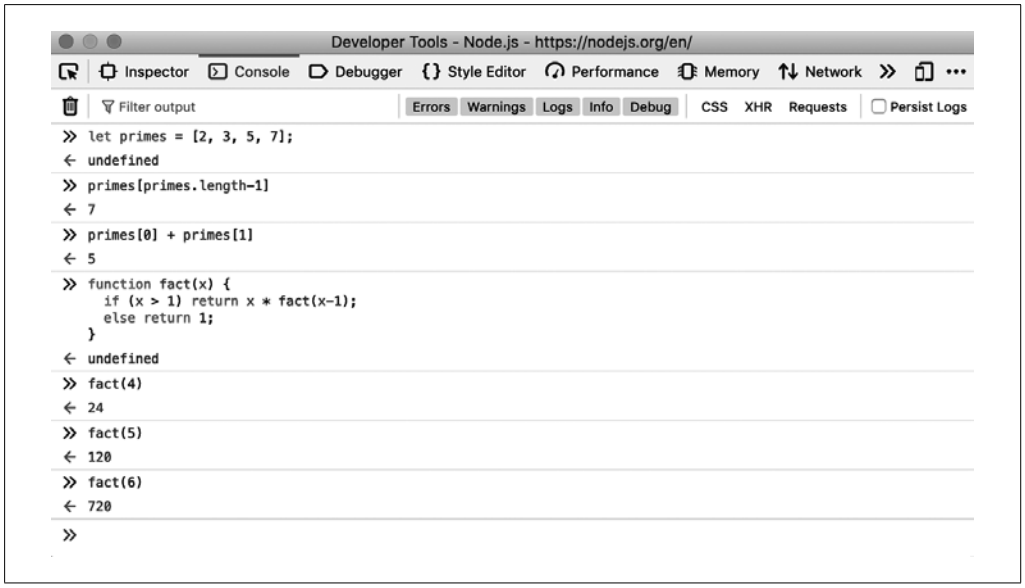
Većina ove knjige fokusirana je na sam jezik JavaScript. Poglavlje 11 dokumentuje JavaScript standardnu biblioteku, poglavlje 15 predstavlja okruženje domaćina veb čitača, a poglavlje 16 predstavlja okruženje kada je domaćin Node.

Ova knjiga prvo pokriva osnove niskog nivoa, a zatim nastavlja na one naprednije i apstrakcije višeg nivoa. Poglavlja su predviđena da se pročitaju više manje po redu. Ali učenje novog programskog jezika nikada nije linearni proces, a ni opis jezika nije linearan: svaka jezička karakteristika povezana je sa drugim osobinama, a ova knjiga je prepuna referenci – ponekad unazad, a ponekad unapred – na srodne materijale. Ovo uvodno poglavlje donosi brzi prvi prolazak kroz jezik, uvodeći ključne karakteristike koje će olakšati razumevanje dubljih tema u narednim poglavljima. Ako već imate praksu kao JavaScript programer, ovo poglavlje verovatno možete preskočiti. (Iako ćete možda uživati čitajući Primer 1-1 na kraju poglavlja pre nego što krenete dalje.)

1.1 Istraživanje JavaScripta

Kada učite novi programski jezik, važno je da isprobate primere iz knjige, a zatim ih modifikujete i izvršite ponovo da biste proverili svoje razumevanje jezika. Da biste to učinili, potreban vam je JavaScript interpreter.

Najlakši način da isprobate nekoliko linija JavaScripta je da otvorite razvojni alat (eng. *developer tools*) u svom veb čitaču (sa F12, Ctrl-Shift-I ili Command-Option-I) i izaberete karticu Console. Zatim, kada se pojavi odzivnik (engl. *promt*), možete otkucati kod i videti rezultate dok kucate. Razvojni alat čitača se pojavljuju kao okna na dnu ili desno od prozora čitača, ali obično ih možete izdvojiti kao odvojene prozore (kao što je prikazano na slici 1-1), što je često prilično zgodno.



Slika 1-1. JavaScript konzola u Firefoxovoj alatki za razvoj

Drugi način da se isproba JavaScript kod je preuzimanje i instaliranje Noda sa *https://nodejs.org*. Jednom kada je Nod instaliran na vašem sistemu, možete jednostavno otvoriti prozor terminala i otkucati **node** da biste započeli interaktivnu JavaScript sesiju poput ove:

```
$ node
Welcome to Node.js v12.13.0.
Type ".help" for more information.
> .help
.break Sometimes you get stuck, this gets you out
.clear Alias for .break
.editor Enter editor mode
.exit Exit the repl
.help Print this help message
.load Load JS from a file into the REPL session
.save Save all evaluated commands in this REPL session to a file

Press ^C to abort current expression, ^D to exit the repl
> let x = 2, y = 3;
undefined
> x + y
5
> (x === 2) && (y === 3)
true
> (x > 3) || (y < 3)
false
```

1.2 Hello World

Kada ste spremni da počnete eksperimentisati sa dužim komadima koda, ova linijska interaktivna okruženja možda više nisu prikladna i verovatno ćete radije da upišete kod u editor teksta. Odatle možete kopirati i prebaciti u JavaScript konzolu ili u sesiju Noda. Ili možete da sačuvate svoj kod u datoteci (tradicionalno tip datoteke za JavaScript kod je *.js*), a zatim pokrenite tu datoteku JavaScript koda pomoću Noda:

```
$ node snippet.js
```

Ako koristite Node na neinteraktivni način kao ovaj, on neće automatski ispisati vrednost celog koda koji pokrenete, tako da ćete to morati sami da uradite. Možete koristiti funkciju `console.log()` da prikažete tekst i ostale JavaScript vrednosti u prozoru vašeg terminala ili u konzoli alata za razvoj. Tako, na primer, ako napravite datoteku *hello.js* koja sadrži ovu liniju koda:

```
console.log ("Hello World!");
```

i izvršite datoteku pomoću `node hello.js`, videćete ispisanu poruku „Hello World!“ (Pozdrav svetu).

Ako želite da se poruka prikazana u JavaScript konzoli, prikaže u veb čitaču, napravite novu datoteku pod nazivom *hello.html* i stavite ovaj tekst u nju:

```
<script src="hello.js"> </script>
```

Zatim učitajte *hello.html* u svoj veb čitač koristeći `file://` URL poput ovog:

```
file:///Users/username/javascript/hello.html
```

Otvorite prozor alata za razvoj da biste videli pozdrav u konzoli.

1.3 Tura kroz JavaScript

Ovaj odeljak predstavlja brzi uvod, kroz primere koda, u JavaScript jezik. Nakon ovog uvodnog poglavlja, zaronimo u JavaScript na najnižem nivou: u drugom poglavlju su objašnjene stvari poput JavaScript komentara, tačaka sa zapetom i skup znakova Unicode. Sa poglavljem 3 počinje biti zanimljivije: objašnjava JavaScript promenljive i vrednosti koje možete dodeliti tim promenljivim.

Evo nekoliko primera koda koji ilustruju ta dva poglavlja:

```
// Bilo šta iza dve kose crte je komentar
// Čitajte komentare pažljivo: objašanjavaju JavaScript kod.

// Promenljiva je simboličeno ime za vrednost.
// Promenljive se deklarišu sa rezervisanom reči let.
let x;                  // Deklarisanje promenjive imena x.
// Vrednosti se dodeljuju promenljivoj sa znakom =
X = 0;                  // Sada promenljiva x ima vrednost 0
x                       // => 0: Promenljiva rezultuje svojoj vrednosti.

// JavaScript podržava nekoliko tipova vrdnosti
x = 1;                  // Brojeve.
x = 0.01;               // Brojevi mogu biti celobrojni ili decimalni.
x = "hello world";      // Niz znakova unutar navodnika.
x = ‘JavaScript’;       // Jednostruki navodnici takođe označavaju niz znakova.
x = true;               // Bulova vrednost.
x = false;              // Druga Bulova vrednost.
x = null;               // Null is a specijalna vrednost koja znači "bez vrednosti."
x = undefined;          // Undefined is druga specijalna vrednost kao i null.
```

Dva druga veoma važna *tipa* kojima JavaScript programi mogu da manipulišu su objekti i nizovi. To su teme iz poglavlja 6 i 7, ali su toliko važne da ćete ih videti mnogo puta pre nego što dođete do tih poglavlja:

```
// Objekat je najvažniji JavaScript tip podataka.
// Objekat je kolekcija parova imena/vrednosti ili niz preslikanih vrijednosti.
let knjiga = {             // Objekti su zatvoreni u vitičaste zagrade.
  tema: "JavaScript",      // Svojstvo "tema" ima vrednost "JavaScript"
  izdanje: 7               // Svojstvo "izdanje" ima vrednost 7
};                         // Vitičasta zagrada označava kraj objekta.

// Pristupite svojstvima objekta pomoću . ili []::
knjiga.tema                // => "JavaScript"
knjiga["izdanje"]          // => 7: another way to access property values.
knjiga.autor = "Flanagan"; // stvaranje novog svojstva dodeljivanjem.
knjiga.sadrzaj = {};       // {} prazan objekat bez svojstava.
```

```
// Uslovno pristupite svojstvima sa ?. (ES2020):
knjiga.sadrzaj?.pog1?.sek1 // => undefined: knjiga.sadrzaj nema svojstvo pog1.

// JavaScript takođe podržava nizove (numerički indeksirane liste) vrednosti:
let prosti = [2, 3, 5, 7]; // Niz od 4 vrednosti, ograničen sa [ i ].
prosti[0]                  // => 2: prvi element (indeks 0) niza.
prosti.length              // => 4: koliko ima elemenata u nizu.
prosti[prosti.length-1]    // => 7: poslednji element nizu.
prosti[4] = 9;             // Dodavanje novog elementa dodelom.
primes[4] = 11;            // Ili mijenjati postojećeg element dodelom.
let empty = [];            // [] je prazan niz bez elemenata.
empty.length               // => 0

// Nizovi i objetki mogu sadržati druge nizove i objekte:
let koordinate = [         // Niz sa 2 elementa.
  {x: 0, y: 0},            // Svaki elenment je jedan objekat.
  {x: 1, y: 1}
];
let data = {               // Objekat sa dva svojstva
  trial1: [[1,2], [3,4]],  // Vrednost svakog svojstva je jedan niz.
  trial2: [[2,3], [4,5]]   // Elementi niza su nizovi.
};
```

Sintaksa komentara u primerima koda

Možda ste u prethodnom kodu primetili da neki od komentara počinju strelicom (`=>`). Oni pokazuju vrednost proizvedenu kodom i moj su pokušaj da u štampanoj knjizi oponašam interaktivno JavaScript okruženja kao što je konzola veb čitača.

Ti `// =>` komentari takođe služe kao *tvrdnja* (engl. *assertion*), i napisao sam alat koji testira kod i proverava da li proizvodi vrednost koja je navedena u komentaru. Ovo bi trebalo da pomogne da se smanji broj grešaka u knjizi.

Postoje dva srodna stila komentara/tvrdnja. Ako vidite komentar oblika `// a == 42`, to znači da će, kada se kod ispred komentara pokrene, varijabla `a` imati vrednost 42. Ako vidite komentar oblika `// !`, To znači da kod ispred komentara daje izuzetak (a ostatak komentara iza uskličnika obično objašnjava kakvu vrstu izuzeća dobijamo).

Videćete ove komentare kroz celu knjigu.

Sintaksa za prikazivanje elemenata niza u uglastim zagradama ili za mapiranje imena svojstava objekata sa vrednostima svojstava unutar vitičastih zagrada poznata je kao izraz inicijalizatora (engl. *initializer expression*), a to je samo jedna od tema poglavlja 4. *Izraz* (engl. *expression*) je fraza JavaScripta koja može biti *izračunata* (engl. *evaluated*) da da vrednost. Na primer, upotreba `.` i `[]` da uputi na vrednost svojstva objekta ili elementa niza.

Jedan od najčešćih načina za formiranje izraza u JavaScriptu je upotreba *operatora*:

```
// Operatori deluju na vrednosti (operande) da bi proizveli novu vrednost.
// Aritmetički operatori su neki od najjednostavnijih:

3 + 2                           // => 5: sabiranje
3 - 2                           // => 1: oduzimanje
3 * 2                           // => 6: množenje
3 / 2                           // => 1.5: deljenje
koordinata[1].x - koordinata[0].x // => 1: nešto složeniji operadni
"3" + "2"                       // => "32": + sabira brojeve, a spaja znakove

// JavaScript definiše neke skrećenice aritmetičkih operacija
let broj = 0;                   // Defisanje promenljive
broj++;                         // Povećava za 1 vrednost promenljive
broj--;                         // Smanjuje za 1 vrednost promenljive
broj += 2;                      // Dodaje 2: isto kao count = broj + 2;
broj *= 3;                      // Množi sa 3: isto kao count = broj * 3;
broj                            // => 6: imena promenljivih su i izrazi

// Operatori za jednakost i relaciju testiraju da li su dve vrednosti jednake,
// nejednake, manje od, veće od, itd. Oni rezultuju sa true (tačno) ili false (netačno).
let x = 2, y = 3;               // Ovi znaci = su dodele, ne test jednakosti
x === y                         // => false: jednakost
x !== y                         // => true: nejednakost
x < y                           // => true: manje od
x <= y                          // => true: manje od ili jednako
x > y                           // => false: veće od
x >= y                          // => false: veće od ili jednako
"dva" === "tri"                 // => false: dva niza znakova se razlikuju
"dva" > "devet"                 // => true: "dv" je alfabetsko veće od "de"
false === (x > y)               // => true: false je jednako sa false

// Logički operatori kombinuju ili menjaju Bulove vredsnosti
(x === 2) && (y === 3)          // => true: oba poređenja su true.
                                // && je AND (logičko I)
(x > 3) || (y < 3)              // => false: ni jedno poređenje nije tačno.
                                // || je OR (logičko ILI)
!(x === y) // => true: ! menja Bulovu vrednost
```

Ako su JavaScript izrazi poput fraza, onda su JavaScript *naredbe* (engl. *statements*) poput rečenica. Naredbe su tema poglavlja 5. Grubo, izraz je nešto što izračunava vrednost, ali ne čini ništa: to ni na koji način ne menja stanje programa. Naredbe, s druge strane, nemaju vrednost, ali menjaju stanje. Gore ste videli deklaracije promenljivih i naredbe za dodeljivanje. Druga široka kategorija naredbi su *kontrolne strukture* (engl. *control structures*), kao što su uslovi i petlje. Videćete primere u nastavku, nakon što obradimo funkcije.

Funkcija (engl. *function*) je imenovani i parametrizovan blok JavaScript koda koji ste definisali jednom, a posle toga se može ponovo i ponovo pozivati. Funkcije nisu obuhvaćene formalno do poglavlja 8, ali poput objekata i nizova, videćete ih mnogo puta pre nego što dođete do tog poglavlja. Evo nekoliko jednostavnih primera:

```
// Funkcije su parametrizovani blokovi JavaScript koda koje možemo pozvati.
function plus1(x) {         // Definišite funkciju nazvanu "plus1" parametrom "x"
    return x + 1;           // Vraća vrednost koja je za 1 veća od primljene vrednosti
}                           // Funkcije se nalaze unutar vitičastih zagrada

plus1(y)                    // => 4: y je 3, tako da ovaj poziv vraća 3+1

let kvadrat = function(x) { // Funkcije su vrednosti i mogu se dodeliti promenljivama
    return x * x;           // Izračunava vrednost funkcije
};                          // tačka sa zapetom označava kraj dodele.

kvadrat(plus1(y))           // => 16: poziva dve funkcije u jednom izrazu
```

U ES6 i u novijim verzijama, postoji kratka sintaksa za definisanje funkcija. Ta sažeta sintaksa koristi => za odvajanje liste argumenata od tela funkcije, tako da su funkcije definisane na ovaj način poznate kao *funkcije strelice* (engl. *arrow functions*). Funkcije strelice najčešće se koriste kada želite da prosledite neimenovanu funkciju kao argument drugoj funkciji. Prethodni kod izgleda ovako kada je prerađen da bi koristio strelice:

```
const plus1 = x => x + 1;   // Ulaz x se mapira u izlaz x + 1
const kvadrat = x => x * x; // Ulaz x se mapira u izlaz x * x
plus1(y)                    // => 4: isti poziv funkcije
kvadrat(plus1(y))           // => 16
```

Kada koristimo funkcije sa objektima, dobijamo *metode*:

```
// Kad se funkcije dodijele svojstvima objekta, nazivamo
// ih "methodama."  Svi JavaScript objekti (uključujući i nizove) imaju methode:
let a = [];                // Pravimo prazan niz
a.push(1,2,3);             // metoda push() dodaje element u niz
a.reverse();               // Druga metoda: preokreće redosled elemenata

// Možemo da definišemo sopstvene metode. Ključna reč "this" odnosi se na objekat
// na kome je metoda definisana: u ovom slučaju ukazuje na niz od ranije.

points.dist = function() { // Definišite metodu za računanje deljne između dve tačke
    let p1 = this[0];      // Prvi element niza na koji se pozvamo
    let p2 = this[1];      // Drugi element objekta "this"
    let a = p2.x-p1.x;     // Razlika u x koordinatama
    let b = p2.y-p1.y;     // Razlika u y koordinatama
    return Math.sqrt(a*a + // Pitagorina teorema
                     b*b); // Math.sqrt() izračunava kvadratni koren
};
points.dist()              // => Math.sqrt(2): distanca između naše 2 tačke
```

Sada, kao što je obećano, evo nekoliko funkcija čija tela pokazuju uobičajene JavaScript naredbe kontrolne strukture:

```
// JavaScript naredbe uključuju uslove i petlje koristeći sintaksu
// od C, C ++, Java i drugih jezika.
function abs(x) {          // Funkcija računa apsolutnu vrednost.
    if (x >= 0) {          // if naredba...
```

```
        return x;          // izvršava ovaj kod ako je poređenje true.
    }                      // Ovo je kraj if uslova.
    else {                 // Opcioni else uslov se izvršava ako je
        return -x;         // poređenje false.
    }                      // Vitičasta zagrada je opciona za jednu naredbu u uslovu.
}                          // return nardba je unutar if/else.
abs(-10) === abs(10)       // => true

function sum(array) {      // Izračunava sumu vrednosti elemenata niza
    let sum = 0;           // Počinje od toga da vrednos sum 0.
    for(let x of array) {  // Ide kroz niz dodeljući svaki element u x.
        sum += x;          // Dodaje vrednost elementa sumi.
    }                      // Ovo je kraj petlje.
    return sum;            // Vraća sumu.
}
sum(primes)                // => 28: suma prvih 5 prostih brojeva 2+3+5+7+11

function factorial(n) {    // Funkcija izračunava faktorijel
    let product = 1;       // Počinje od toga da je product 1
    while(n > 1) {         // Ponavlja naredbu u {} dok je (while) izraz u () true
        product *= n;      // Skraćeni oblik za product = product * n;
        n--;               // Skraćeni oblik za n = n - 1
    }                      // Ovo je kraj petlje
    return product;        // Vraća product
}
factorial(4)               // => 24: 1*4*3*2

function factorial2(n) {   // Drgua verzija sa drugačijom petljom
    let i, product = 1;    // Počinje od 1
    for(i=2; i <= n; i++)  // Automatski povećava i do 2 do n
        product *= i;      // Radi ovo svaki put. {} nisu potrebne za jednoredu petlju
    return product;        // Vraća factorijel
}
factorial2(5)              // => 120: 1*2*3*4*5
```

JavaScript podržava objektno orijentisani stil programiranja, ali se značajno razlikuje od „klasičnih“ objektno orijentisanih programskih jezika. Poglavlje 9 detaljno pokriva objektno orijentisano programiranje u JavaScriptu, sa puno primera. Evo vrlo jednostavnog primera koji pokazuje kako definisati JavaScript klasu koja će predstavljati 2D geometrijske tačke. Objekti koji predstavljaju instance ove klase imaju jedinstvenu metodu, nazvanu `distance()`, koja izračunava udaljenost tačke od početka:

```
class Point {              // Po konvenciji, imena klasa počinju velikim slovom.
    constructor(x, y) {    // Funkcija constructor za inicijalizaciju novih instanci.
        this.x = x;        // Ova ključna reč je objekat koji se inicijalizuje.
        this.y = y;        // Smešta argmente funkcije kao svojstva objekta.
    }                      // Ne treba return u funkciji konstruktoru

    distance() {           // Method izračunava distancu od početka do tačke.
        return Math.sqrt(  // Vraća kvadrati koren od x² + y².
            this.x * this.x +  // this se odnosi na objekat Point object za koji
```

```
            this.y * this.y    // se poziva metoda distance.
        );
    }
}

// Upotrebite funkciju constructor Point() sa "new" da biste kreirali Point objekte
let p = new Point(1, 1);   // Geometrijska tačka (1,1).

// Sada upotrebite metodu p objekta Point
p.distance()               // => Math.SQRT2
```

Ovde se završava uvodna turneja po osnovnoj sintaksi i mogućnostima JavaScripta, ali knjiga se nastavlja sa samostalnim poglavljima koja pokrivaju dodatne funkcije jezika:

Poglavlje 10, Moduli

Prikazuje kako JavaScript kod u jednoj datoteci ili skripti može koristiti JavaScript funkcije i klase definisane u drugim datotekama ili skriptama.

Poglavlje 11, Standardna biblioteka JavaScripta

Obuhvata ugrađene funkcije i klase koje su dostupne svim JavaScript programima. Ovo uključuje važne strukture podataka kao što su mape i skupovi, klase regularnih izraza za upoređivanje tekstualnih uzoraka, funkcije za serializaciju JavaScript strukrutra podataka i mnogo više.

Poglavlje 12, Iteratori i generatori

Objašnjava kako radi `for/of` petlja i kako možete napraviti sopstvene iterabilne klase `for/of`. Takođe obuhvata funkcije generatora i naredbu o prinosu.

Poglavlje 13, Asinhroni JavaScript

Ovo poglavlje je detaljno istraživanje asinhronog programiranja u JavaScript-u, obuhvata povratne pozive i događaje, API-e zasnovane na obećanjima i ključne reči asinhronizacije i čekanja. Iako jezgro JavaScript jezika nije asinhrono, asinhroni API-ji su zadati i veb čitačima i u Nodu, a ovo poglavlje objašnjava tehnike rada sa tim API-jima.

Poglavlje 14, Metaprogramiranje

Predstavlja brojne napredne funkcije JavaScripta koje mogu biti zanimljive programerima koji pišu biblioteke koda za upotrebu od strane drugih JavaScript programera.

Poglavlje 15, JavaScript u veb čitačima

Uvodi okruženje veb čitača, objašnjava kako veb čitači izvršavaju JavaScript kod i pokriva najvažniji od mnogih API-ja definisanih od strane veb čitača. Ovo je daleko najduže poglavlje u knjizi.

Poglavlje 16, JavaScript na strani servera i Node

Uvodi okruženje Noda, a pokriva osnovni programski model i strukture podataka i API-je koji su najvažniji za razumevanje.

Poglavlje 17, JavaScript alati i proširenja

Pokriva alate i jezične nastavke o kojima vredi znati jer se široko koriste i mogu vas učiniti produktivnijim programerom.

1.4 Primer: Histogrami učestalosti znakova

Ovo poglavlje se završava kratkim, ali netrivijalnim JavaScript programom. Primer 1-1 je Node program koji čita tekst sa standardnog unosa, izračunava histogram frekvencije znakova iz unetog teksta, a zatim ispisuje histogram. Možete pozvati ovaj program da analizirate frekvenciju znakova svog izvornog koda:

```
$ node charfreq.js < charfreq.js
T: ########### 11.22%
E: ########## 10.15%
R: ####### 6.68%
S: ###### 6.44%
A: ###### 6.16%
N: ###### 5.81%
O: ##### 5.45%
I: ##### 4.54%
H: #### 4.07%
C: ### 3.36%
L: ### 3.20%
U: ### 3.08%
/: ### 2.88%
```

Ovaj primer koristi brojne napredne JavaScript funkcije i namenjen je da prikaže kako mogu izgledati JavaScript programi u stvarnom svetu. Ne bi trebalo očekivati da ćete razumeti sav kod, ali budite sigurni da će sve to biti objašnjeno u narednim poglavljima.

Primer 1-1. Izračunavanje histograma učestalosti znakova pomoću JavaScripta

```
/**
* Ovaj Node program čita tekst sa standardnog ulaza, izračunava frekvenciju
* svakog slova u tom tekstu, i prikazuje histograma najviše
* korišćenih znakovi. Za pokretanje je potreban Node 12 ili noviji.
*
* U okruženju tipa Unix možete pozvati program ovako:
*   node charfreq.js < corpus.txt
* /

// Ova klasa proširuje Map tako da metoda get () vraća zadatu
// vrednost umesto null kada indeks (key) nije na mapi
class DefaultMap extends Map {
  constructor(defaultValue) {
    super();                           // poziva konstruktor superclass
    this.defaultValue = defaultValue; // pamti unapred zadatu vrednost
  }
```

```
  get(key) {
    if (this.has(key)) {              // ako je indeks key u mapi
    return super.get(key);            // vraća njegovu vrednost iz superklase.
  }
  else {
    return this.defaultValue;         // U protivnom vraća zadatu vrednost
  }
}

}
// Ova klasa izračunava i prikazuje histogram učestalosti znakova
class Histogram {
  constructor() {
    this.letterCounts = new DefaultMap(0); // mapira znakove u broj
    this.totalLetters = 0;                 // koliko ima ukupno slova
  }

  // Ova funkcija ažurira histogram slovima iz teksta
  add(text) {
    // Uklanjamo duple prazne znakove iz teksta, i pretvaramo u velika slova
    text = text.replace(/\s/g, "").toUpperCase();

    // Sada prolazimo kroz znakove teksta
    for(let character of text) {
      let count = this.letterCounts.get(character); // Stara vrednost koliko ih ima
      this.letterCounts.set(character, count+1);    // Povećavamo
      this.totalLetters++;
    }
  }

  // Pretvara histogram u niz znakova da prikaže ASCII grafiku
  toString() {
    // Pretvara Map u niz [key,value] nizova
    let entries = [...this.letterCounts];

    // Sortira niz po broju, a zatim abecedno
    entries.sort((a,b) => {           // Funkcija za sort redosled
      if (a[1] === b[1]) {            // Ako su brojevi isti
        return a[0] < b[0] ? -1 : 1; // sortiraj abecedno.
      } else {                        // Ako se brojevi razlikuju
        return b[1] - a[1];           // sortiraj po većem broju
      }
    });

    // Pretvara brojeve u procente
    for(let entry of entries) {
      entry[1] = entry[1] / this.totalLetters*100;
    }

    // Odbacuje vrednosti manje od 1%
    entries = entries.filter(entry => entry[1] >= 1);
```

```
    // Pretvara svaku stavku u liniju teksta
    let lines = entries.map(
      ([l,n]) => `${l}: ${"#".repeat(Math.round(n))} ${n.toFixed(2)}%`
    );

    // Vraća spojene linije, razdvojene znakom za novi red.
    return lines.join("\n");
  }
}


// Ova asihrona funkcija (vraća obećanje) stvara objekat Histogram,
// asinhrono čita delova teksta sa standardnog ulaza i dodaje te komade u
// histogram. Kada dođe do kraja podataka, vraća ovaj histogram
async function histogramFromStdin() {
  process.stdin.setEncoding("utf-8"); // Čita Unicode nizove znakove, ne bajtove
  let histogram = new Histogram();
  for await (let chunk of process.stdin) {
    histogram.add(chunk);
  }
  return histogram;
}

// Ova poslednja linija koda je glavno telo programa.
// Stvara objekat Histogram sa standardnog ulaza, a zatim prikazuje histogram.
histogramFromStdin().then(histogram => { console.log(histogram.toString()); });
```

1.5 Rezime

Ova knjiga objašnjava JavaScript odozdo nagore. To znači da počinjemo sa detaljima niskog nivoa poput komentara, identifikatora, promenljivih i tipova; zatim nadograđujemo izraze, naredbe, objekte i funkcije; a zatim pokrivamo jezičke apstrakcije visokog nivoa poput klasa i modula. Reč *sveobuhvatan* u naslovu ove knjige shvatam ozbiljno, a naredna poglavlja objašnjavaju jezik na nivou detalja koji u početku može biti neugodan. Međutim, istinsko savladavanje JavaScripta zahteva razumevanje detalja i nadam se da ćete imati vremena da pročitate ovu knjigu od korica do korica. Ali molim vas nemojte osećati da to morate da učinite pri prvom čitanju. Ako se nađete zaglavljeni u nekom delu, jednostavno pređite na sledeći. Možete se vratiti i savladati detalje nakon što steknete znanje jezika u celini.

POGLAVLJE 2

Leksička struktura

Leksička struktura programskog jezika je skup elementarnih pravila koja određuje kako pišete programe na tom jeziku. To je sintaksa jezika na najnižem nivou: određuje kako izgledaju imena promenljivih, znakovi razgraničenja za komentare i kako se jedna programska naredba odvaja od sledeće. Ovo kratko poglavlje dokumentuje leksičku strukturu JavaScripta. Pravilo obuhvata:

- Razlikovanje veličine slova, razmaka i kraja linija
- Komentare
- Literale
- Identifikatore i rezervisane reči
- Unicode
- Opciono tačku sa zapetom

2.1 Tekst JavaScript programa

JavaScript je jezik koji razlikuje velika i mala slova. To znači da rezervisane reči jezika, promenljive, imena funkcija i drugi *identifikatori* jezika uvek moraju biti napisani sa doslednom veličinom slova. Na primer, rezervisana reč (engl. *keyword*) `while` mora da se upiše „while„, a ne „While„ ili „WHILE„. Slično tome, `online`, `Online`, `OnLine` i `ONLINE` su četiri različita imena promenljivih.

JavaScript zanemaruje razmake koji se pojavljuju između tokena, tj. leksema (engl. *token*) u programima. JavaScript takođe uglavnom ignoriše krajeve redova (ali za izuzetak pogledajte §2.6). Budući da znakove za razmak i znakove za novi red možete slobodno koristiti u svojim programima, možete ih upotrebiti da formatirate i uvlačete redove na uredan i dosledan način što olakšava čitanje i razumevanje koda.

Pored znaka za razmak (`\u0020`), JavaScript prepoznaje i tabove, razne ASCII kontrolne znakove i razne znakove Unicoda kao znak za razmak. JavaScript prepoznaje nove redove, povratak na početak reda (engl. *carriage returns*) i par povratak na početak reda/novi red (engl. *carriage return/line feed*) kao završetak reda.

2.2 Komentari

JavaScript podržava dva stila komentara. Bilo koji tekst između // i kraja reda tretira se kao komentar i JavaScript ga ignoriše. Bilo koji tekst između znakova /* i */ takođe se tretira kao komentar; ovi komentari mogu da obuhvataju više linija, ali ne mogu da se ugnezde. Sledeći redovi koda su sve važeći JavaScript komentari:

```
// Ovo je komenatar u jednom redu

/* I ovo je komentar */

/*
 * Ovo je višeredni komentar. Dodatni znak * nije potreban na početku
 * svakog reda, ali izgleda lepše!
 */
```

2.3 Literali

Literal je doslovna vrednost koja se pojavljuje direktno u programu. Slede svi literali:

```
12              // Broj dvanaest
1.2             // Broj jedan tačka dva (jedan zapeta dva)
"Zdravo svima"  // Niz znakova
'Hej'           // Drugi niz znakova
True            // Bulova vrednost
False           // Druga Bulova vrednost
Null            // Prazno, bez objekta
```

Kompletni detalji o numeričkim i znakovnim literalima nalaze se u poglavlju 3.

2.4 Identifikatori i rezervisane reči

Identifikator (engl. *identifier*) je jednostavno ime. U JavaScriptu se identifikatori koriste za imenovanje konstanti, promenljivih, svojstava, funkcija i klasa i za obezbeđivanje oznaka za određene petlje u JavaScript kodu. JavaScript identifikator mora početi slovom, znakom podvlačenja (_) ili znakom dolara ($). Naredni znakovi mogu biti slova, cifre, znaci podvlačenja ili oznaka dolara. (Brojke nisu dozvoljene kao prvi znak, tako da JavaScript može lako razlikovati identifikatore od brojeva.) Sve su to legalni identifikatori:

```
i
ime_moje_variable
v13
_ostalo
$str
```

Kao i svaki jezik, JavaScript zadržava određene identifikatore za sopstvenu upotrebu. Ove „rezervisane reči“ ne mogu se koristiti kao važeći identifikatori. Oni su navedeni u sledećem odeljku.

2.4.1 Rezervisane reči

Sledeće reči su deo JavaScript jezika. Mnoge od ovih (kao što su `if`, `while` i `for`) su rezervisane reči koje se ne smeju koristiti kao imena konstanti, promenljivih, funkcija ili klasa (mada se mogu koristiti kao nazivi svojstava unutar objekta). Ostali (kao što su `from`, `of`, `get` i `set`) se koriste u ograničenom kontekstu bez sintaktičke nejasnoće i savršeno su važeći kao identifikatori. Ostale ključne reči (poput `let`) ne mogu se u potpunosti rezervisati da bi se zadržala kompatibilnost unazad sa starijim programima, pa postoje složena pravila koja regulišu kada se mogu koristiti kao identifikatori i kada ne mogu. (`let` se koristi kao naziv promenljive ako se, na primer, deklariše sa `var` izvan klase, ali ne i ako je deklarisano unutar klase ili sa `const`.) Najjednostavniji kurs je izbegavanje korišćenja bilo koje od ovih reči kao identifikatora, osim `from`, `set` i `target`, koji su bezbedni za upotrebu i koji su već u uobičajenoj upotrebi.

```
as      const     export    get         null    target   void
async   continue  extends   if          of      this     while
await   debugger  false     import      return  throw    with
break   default   finally   in          set     true     yield
case    delete    for       instanceof  static  try
catch   do        from      let         super   typeof
class   else      function  new         switch  var
```

JavaScript takođe zadržava ili ograničava upotrebu određenih ključnih reči koje jezik trenutno ne koristi, ali koje bi se mogle koristiti u budućim verzijama:

```
enum  implements  interface  package  private  protected  public
```

Iz istorijskih razloga, `arguments` i `eval` nisu dozvoljeni kao identifikatori u određenim okolnostima i najbolje ih je u potpunosti izbegavati.

2.5 Unicode

JavaScript programi su napisani pomoću Unicode skupa znakova, a možete koristiti bilo koje Unicode znakove u znakovnim nizovima i komentarima. Za prenosivost i lakoću uređivanja, uobičajeno je da se u identifikatorima koriste samo ASCII slova i cifre. Ali ovo je samo programska konvencija i jezik omogućava Unicode slova, cifre i ideografije (ali ne i emoje) u identifikatorima. To znači da programeri mogu koristiti matematičke simbole i reči iz neengleskih jezika kao konstante i promenljive:

```
const π = 3.14;
const sí = true;
```

2.5.1 Escape Unicode nizovi

Neki računarski hardver i softver ne može prikazati, uneti ili pravilno obraditi kompletan skup Unicode znakova. Da bi podržao programere i sisteme koji koriste stariju tehnologiju, JavaScript definiše Esc sekvence koje nam omogućavaju da pišemo Unicode znakove koristeći samo ASCII znakove. Ovi Unicode nizovi započinju znakom `\u` i prate ga tačno četiri heksadecimalne cifre (koristeći velika ili mala slova A – F) ili jedna do šest heksadecimalnih

cifara unutar vitičastih zagrada. Ovi Unicode nizovi mogu se pojaviti u JavaScript literalima, regularnim izrazima i identifikatorima (ali ne u rezervisanim rečima jezika). Unicode niz za znak „é“, na primer, je `\u00E9`; evo tri različita načina za pisanje imena promenljive koja uključuju ovaj znak:

```
let café = 1; // Definisanje promenljive pomoću Unicode znaka
caf\u00e9     // => 1; pristup promenljivoj pomoću escape niza
caf\u{E9}     // => 1; drugi oblik istog escape niza
```

Rane verzije JavaScripta podržale su samo četvorocifreni escape nizove. Verzija s vitičastima zagradama uvedena je u ES6 kako bi bolje podržao Unicode za koji je potrebno više od 16 bita, poput emojija:

```
console.log("\u{1F600}");  // Daje emoji nasmejano lice
```

Escape Unicode nizovi mogu se pojaviti i u komentarima, ali pošto se komentari zanemaruju, u tom kontekstu se jednostavno tretiraju kao ASCII znakovi i ne tumače se kao Unicode.

2.5.2 Normalizacija Unicoda

Ako koristite ne-ASCII znakove u svojim JavaScript programima, morate biti svesni da Unicode dopušta više od jednog načina kodiranja istog znaka. Znak „é“, na primer, može se kodirati kao jedinstveni Unicode znak `\u00E9` ili kao običan ASCII „e“, praćen znakom akutnog akcenta koji kombinuje znak `\u0301`. Ova dva kodiranja obično izgledaju potpuno isto kada ih prikaže editor teksta, ali imaju različite binarne kodove, što znači da ih JavaScript smatra različitim, što može dovesti do vrlo zbunjujućih programa:

```
const café = 1;  // Ova konstanta se zove "caf\u{e9}"
const café = 2;  // Ovo je drugačija konstanta: "cafe\u{301}"
café             // => 1: ova konstanta ima jednu vrednost
café             // => 2: ova, iako se ne razlikuje, ima drugu vrednost
```

Unicode standard definiše preferisano kodiranje svih znakova i određuje postupak normalizacije za pretvaranje teksta u kanonski oblik pogodan za upoređivanje. JavaScript podrazumeva da je kod koji tumači već normiran i *ne vrši* nikakvu svoju normalizaciju. Ako planirate da koristite Unicode znakove u svojim JavaScript programima, trebalo bi da obezbedite da vaš editor ili neki drugi alat izvodi Unicode normalizaciju vašeg izvornog koda kako biste sprečili da završite sa različitim, ali vizuelno istim identifikatorima.

2.6 Neobavezni znaci tačka sa zapetom

Kao i mnogi programski jezici, JavaScript koristi znak tačka sa zapetom (engl. *semicolon*) (`;`) za odvajanje naredbi (vidi poglavlje 5) jedne od druge. Ovo je važno za postizanje jasnog značenja vašeg koda: bez znaka razdvajanja, kraj jedne naredbe može se činiti početkom sledeće, ili obratno. U JavaScriptu obično možete izostaviti tačku sa zapetom između dve naredbe ako su te naredbe napisane u zasebnim redovima. (Takođe možete izostaviti tačku sa zapetom na kraju programa ili ako je sledeći token u programu zatvarajuća vitičasta zagrada: `}`.) Mnogi JavaScript programeri (i kod u ovoj knjizi) koriste tačku sa zapetom da izričito obeleže krajeve naredbi, čak i tamo gde nije potrebno. Drugi stil je izostavljanje tačke sa

zapetom kad god je to moguće, koristeći ih samo u retkim situacijama. Bez obzira koji stil izaberete, treba da razumete nekoliko detalja o neobaveznim znacima tačka sa zapetom u JavaScriptu.

Razmotrite sledeći kod. Obzirom da se dve naredbe pojavljuju u zasebnim redovima, prva tačka sa zapetom se može izostaviti:

```
a = 3;
b = 4;
```

Napisano ovako, prva tačka sa zapetom je potrebna:

```
a = 3; b = 4;
```

Imajte na umu da JavaScript ne tretira svaki kraj reda kao tačku sa zapetom: obično tretira kraj reda kao tačku sa zapetom samo ako ne može da raščlani (engl. *parse*) kod bez dodavanja implicitnog znaka tačka sa zapetom. Formalnije (i uz tri izuzetaka opisana malo kasnije), JavaScript tretira kraj reda kao tačku sa zapetom ako se sledeći znak koji nije prazan ne može protumačiti kao nastavak trenutne naredbe. Razmotrite sledeći kod:

```
let a
a
=
3
console.log(a)
```

JavaScript tumači ovaj kod ovako:

```
let a; a = 3; console.log(a);
```

JavaScript tretira prelom prvog reda kao tačku sa zapetom jer ne može raščlaniti kod `let a` bez tačke sa zapetom. Drugi `a` može stajati samo kao naredba `a;`, ali JavaScript ne tretira drugi kraj red kao tačku sa zapetom jer može nastaviti sa raščlanjivanjem duže naredbe `a = 3;`.

Ova pravila o kraju naredbi dovodi do nekih iznenađujućih slučajeva. Ovaj kod izgleda kao dve odvojene naredbe odvojene novim redom:

```
let y = x + f
(a+b).toString()
```

Zagrade u drugom redu koda mogu se protumačiti kao poziv funkcije `f` iz prve linije, a JavaScript interpretira kod ovako:

```
let y = x + f(a+b).toString();
```

Verovatno ovo nije tumačenje koje je autor koda hteo. Da bi ovo radilo kao dve odvojene naredbe, u ovom slučaju je potrebna eksplicitna tačka sa zapetom.

Generalno, ako naredba započinje sa `(`, `[`, `/`, `+` i `-`, postoji šansa da se ona može protumačiti kao nastavak prethodne naredbe. Naredbe koje počinju sa `/`, `+` i `–` su u praksi prilično retke, ali naredbe koje počinju sa `(` i `[` uopšte nisu neuobičajene, bar u nekim stilovima JavaScript programiranja. Neki programeri vole da stave odbrambenu tačku sa zapetom na početak svake takve naredbe tako da će kod i dalje raditi ispravno čak i ako je prethodna naredba promenjena i prethodna tačka sa zapetom uklonjena:

```
let x = 0                          // tačka i zapeta izostavljeni ovde
;[x,x+1,x+2].forEach(console.log) // Odbrambena ; čini naredbu odvojenu
```

Postoje tri izuzetka od opšteg pravila da JavaScript interpretira krajeve redova kao tačku sa zapetom, kada drugi red ne može analizirati kao nastavak naredbe u prethodnom redu. Prvi izuzetak uključuje `return`, `throw`, `yield`, `break` i `continue` naredbe (vidi poglavlje 5). Ove naredbe često su samostalne, ali ih ponekad prati identifikator ili izraz. Ako se posle bilo koje od ovih reči (pre bilo kojeg drugog tokena) pojavi kraj reda, JavaScript će uvek kraj reda shvatiti kao tačku sa zapetom. Na primer, ako napišete:

```
return
true;
```

JavaScript smatra da mislite:

```
return; true;
```

A vi ste možda hteli:

```
return true;
```

To znači da ne smete umetati kraj reda između `return`, `break` ili `continue` i izraza koji prati ključnu reč. Ako umetnete kraj reda, verovatno će se kod izvršti na neočigledan način koji je teško shvatiti.

Drugi izuzetak uključuje operatore `++` i `--` (§4.8). Ovi operatori mogu biti operatori prefiksa koji se pojavljuju pre izraza ili postfiks operatori koji se pojavljuju posle izraza. Ako želite da koristite bilo koji od ovih operatora kao postfiks operatore, oni se moraju pojaviti u istoj liniji kao i izraz na koji se odnose. Treći izuzetak uključuje funkcije definisane pomoću sažete sintakse „strelica“: => strelica se mora pojaviti u istom redu kao i lista parametara.

2.7 Rezime

Ovo poglavlje pokazalo je kako se JavaScript programi pišu na najnižem nivou. Sledeće poglavlje nas vodi jedan korak više i uvodi osnovne tipove i vrednosti (brojevi, znakovne nizove i tako dalje) koji služe kao osnovne jedinice izračunavanja u JavaScript programima.